



# **IO-Link Master LLD**

## **Software Design**

**Version 0.1**

**Business Unit:**

**Project Name: TI PDK**

**TABLE OF CONTENTS**

<b>1. PURPOSE .....</b>	<b>5</b>
<b>2. FUNCTIONAL OVERVIEW .....</b>	<b>5</b>
<b>3. ASSUMPTIONS .....</b>	<b>5</b>
<b>4. DEFINITIONS, ABBREVIATIONS, ACRONYMS .....</b>	<b>5</b>
<b>5. REFERENCES .....</b>	<b>6</b>
<b>6. DESIGN CONSTRAINTS .....</b>	<b>6</b>
6.1 EXTERNAL CONSTRAINTS / FEATURES .....	6
6.2 EXTERNAL CONSTRAINTS / SYSTEM PERFORMANCE .....	6
6.3 INTERNAL CONSTRAINTS / REQUIREMENTS .....	6
<b>7. SYSTEM OVERVIEW .....</b>	<b>6</b>
7.1 IO-LINK MASTER LLD REQUIREMENTS .....	6
7.2 SYSTEM CONTEXT .....	7
7.3 FUNCTIONAL DESCRIPTION .....	7
7.3.1 <i>Common Software IP driver</i> .....	8
7.3.1.1 <i>IOLINK_open API</i> .....	9
7.3.1.2 <i>IOLINK_close API</i> .....	9
7.3.1.3 <i>IOLINK_control API</i> .....	9
7.3.2 <i>IO-Link Master Timers</i> .....	9
7.3.3 <i>IO-Link master interrupts</i> .....	11
7.3.3.1 <i>Hardware Interrupts</i> .....	11
7.3.3.2 <i>Software Interrupts and callbacks</i> .....	12
7.3.4 <i>IO-Link Master Call Flows</i> .....	12
7.4 INTEGRATION WITH IO-LINK MASTER STACK .....	16
7.5 SYSTEM RESOURCES .....	18
<b>8. TEST PLAN .....</b>	<b>18</b>
8.1 HARDWARE .....	18
8.2 TEST SETUP .....	19
8.3 TEST AUTOMATION .....	19
<b>9. STANDARDS, CONVENTIONS AND PROCEDURES .....</b>	<b>19</b>
9.1 DOCUMENTATION STANDARDS .....	19
9.2 NAMING CONVENTIONS .....	19

9.3	PROGRAMMING STANDARDS .....	19
9.4	SOFTWARE DEVELOPMENT TOOLS .....	20
<b>10.</b>	<b>SYSTEM DESIGN .....</b>	<b>20</b>
10.1	DESIGN APPROACH .....	20
10.2	DEPENDENCIES .....	20
<b>11.</b>	<b>IP FEATURE LIST COMPARISON .....</b>	<b>21</b>

## TABLE OF FIGURES

Figure 1 : Process SDK driver subsystem architecture.....	7
Figure 2 : IO-LINK configuration code flow .....	<b>Error! Bookmark not defined.</b>
Figure 3 : IO-LINK read operation in blocking mode.....	<b>Error! Bookmark not defined.</b>
Figure 4 : IO-LINK read operation in callback mode .....	<b>Error! Bookmark not defined.</b>
Figure 5 : IO-LINK write operation in blocking mode .....	<b>Error! Bookmark not defined.</b>
Figure 6 : IO-LINK write operation in callback mode .....	<b>Error! Bookmark not defined.</b>
Figure 7 : IO-LINK EDMA configuration sequence.....	<b>Error! Bookmark not defined.</b>
Figure 8 : IO-LINK read sequence with EDMA .....	<b>Error! Bookmark not defined.</b>
Figure 9 : IO-LINK write operation sequence with EDMA.....	<b>Error! Bookmark not defined.</b>
Figure 10 : IO-LINK Interrupt handling function .....	<b>Error! Bookmark not defined.</b>
Figure 10 : IO-LINK LLD Subsystem Block Diagram .....	<b>Error! Bookmark not defined.</b>

**Revision Control**

<b>Author Name</b>	<b>Description</b>	<b>Version</b>	<b>Date</b>
Hao Zhang	Initial version	0.1	Oct 22 <sup>nd</sup> 2018

## 1. Purpose

This document describes the functionality, architecture, and operation of the IO-Link Master Low Level Driver. Also, the data types, data structures and application programming interfaces (APIs) provided by the IO-LINK MASTER driver are explained in this document.

## 2. Functional Overview

The IO-LINK MASTER driver provides an interface to any IO-Link devices (sensors or actuators) accessible via a half-duplex UART serial bus on SoC. It also provides a well-defined API layer which allows applications to manage IO-Link devices via simple portable APIs.

## 3. Assumptions

NA

## 4. Definitions, Abbreviations, Acronyms

<b>Term</b>	<b>Definition</b>
API	Application Programming Interface
CSL	Chip Support Library
LLD	Low Level Driver
MMR	Memory Mapped Registers
OSAL	Operating System Adaptation Layer
IP	Intellectual Property
SOC	System On Chip
STDIO	Standard Input/Output
PRU-ICSS	Programmable Realtime Unit – Industrial Communication Subsystem

## 5. References

The following references are related to the features described in this document and shall be consulted as necessary.

- TRM of the SoCs being supported by the IO-LINK MASTER LLD
- IO-Link Consortium, [IO-Link Interface and System Specification Version 1.1.2](#)
- [TI Eight Port IO-Link Master Reference Design](#)

## 6. Design Constraints

### 6.1 External Constraints / Features

- IO-LINK MASTER LLD should access OS components only through OSAL.

### 6.2 External Constraints / System Performance

- None

### 6.3 Internal Constraints / Requirements

IO-LINK MASTER LLD should use CSL layer for register access to abstract the HW dependencies and maintain portability across the platforms.

## 7. System Overview

### 7.1 IO-Link Master LLD Requirements

- Enable IO Link Master on PRU-ICSS marketing requirement ([CATREQ-2053](#))
  - Support 8 data channels ([PRSDK-4866](#))
  - Support 8-bit data length ([PRSDK-4865](#))
  - Support 1start bit and 1 stop bit ([PRSDK-4864](#))
  - Support COM3/COM2/COM1 baud rate ([PRSDK-4862](#))
  - Enable IO Link Master on PRU-ICSS ([PRSDK-3401](#))
  - Support even parity bit check ([PRSDK-4863](#))
  - Support channel independent time control and monitoring ([PRSDK-4851](#))
  - Support test example which integrated with an IO-Link master stack ([PRSDK-4867](#))
  - Support software interrupt APIs in OSAL ([PRSDK-4858](#))
  -

**7.2 System Context**

IO-LINK MASTER LLD is designed to be functional as part of TI processor SDK driver package. There will be several components in the processor SDK, apart from applications, which uses IO-LINK MASTER LLD. Driver design ensures that it fits into system properly and provides suitable APIs for utilizing the ICSS PRU Message Handler for IO-Link Master.

The following figure shows the architecture of processor SDK sub-system around the LLD modules.

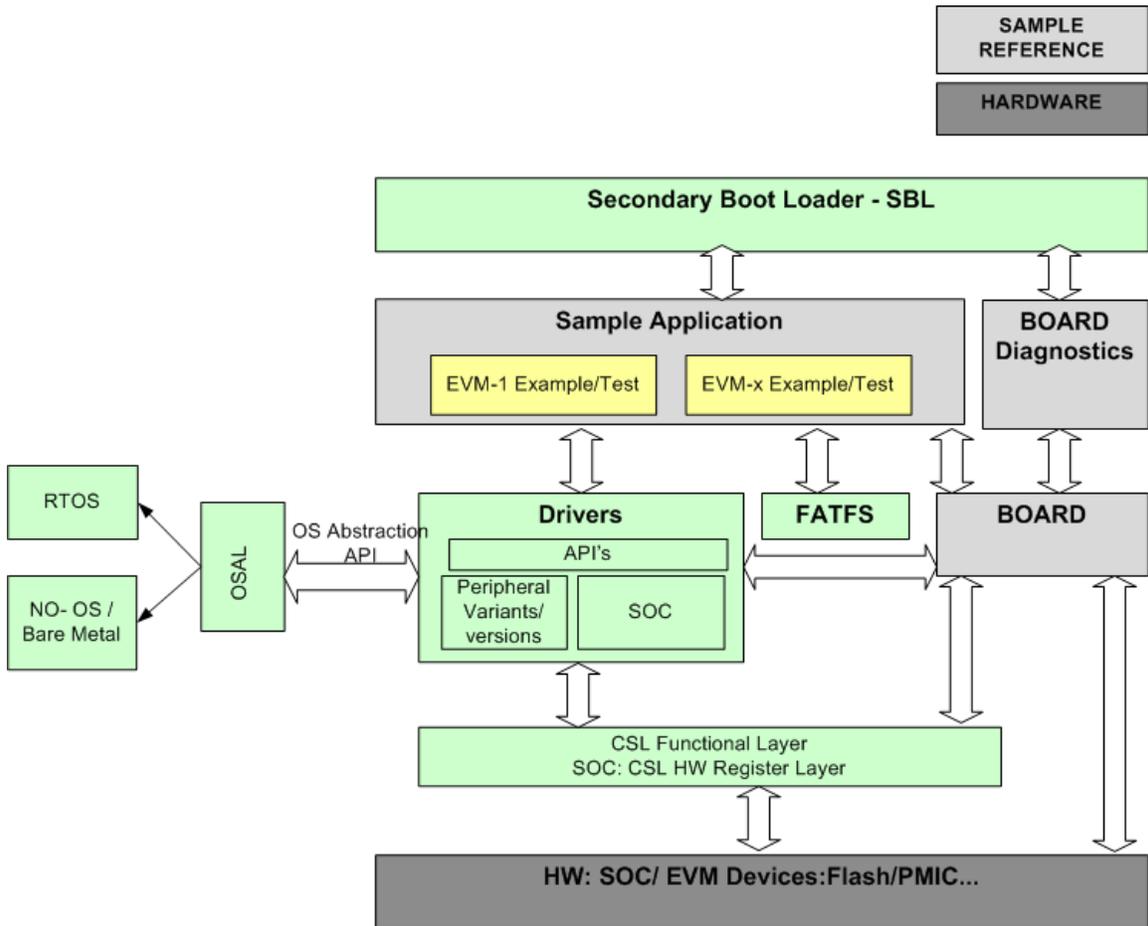


Figure 1 : Process SDK driver subsystem architecture

**7.3 Functional Description**

The following figure shows the block diagram of the IO-link Master application.

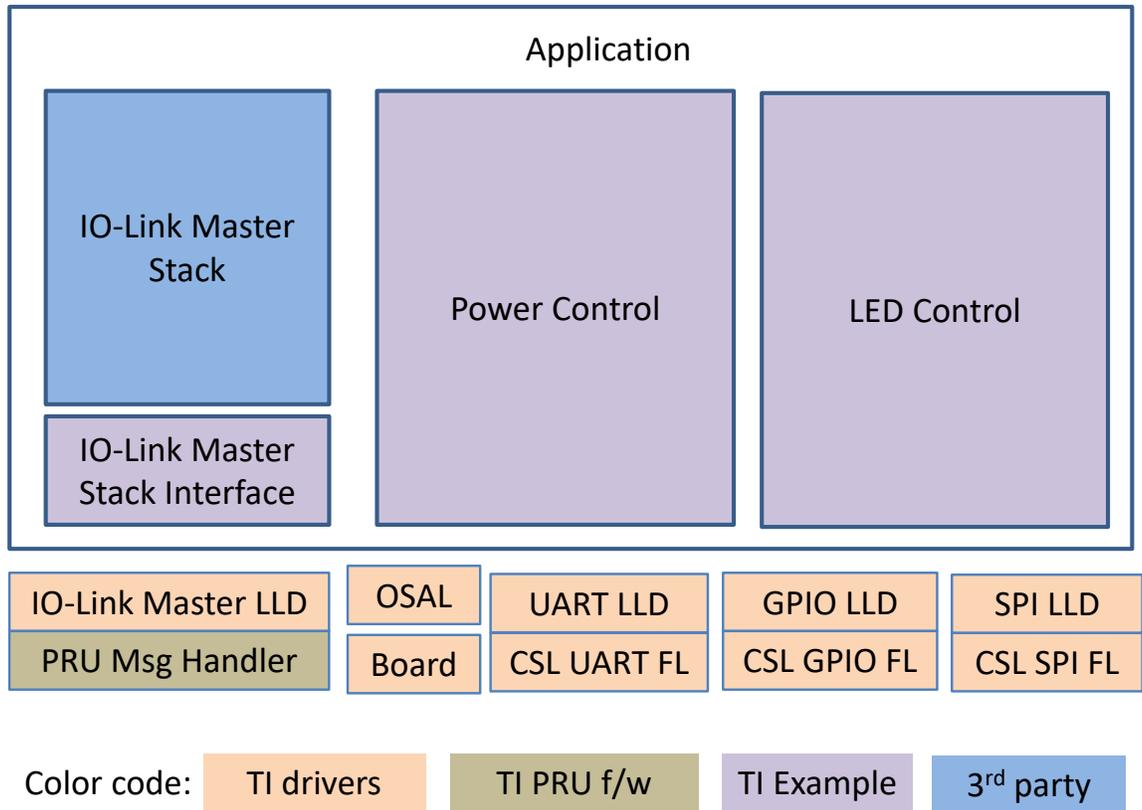


Figure 2 : IO-Link Master Application Block Diagram

IO-LINK MASTER LLD supports the following major functionalities as described below

- Standard APIs to initialize/de-initialize/control the common Software IP driver
- Configuration/Control of the PRU firmware to handle the messages transferred between the IO-Link master and slave
- Timer/Interrupt driven IO-Link communication between master and slave to meet the timing requirement of IO-Link master protocol
- SoC specific driver which can be easily ported across different platforms

### 7.3.1 Common Software IP driver

The common software IP driver provides the following APIs to initialize/de-initialize/control an IO-Link master instance.

- IOLINK\_open
- IOLINK\_close
- IOLINK\_control

### 7.3.1.1 IOLINK\_open API

The open API performs the following functions when called by the application:

- open an IO-Link master instance which supports up to 8 ports (channels) with the IO-Link devices
- register the hardware and software interrupts
- initialize the PRU
  - load the firmware to PRU instruction memory
  - load the configuration data (LUT, etc.) to PRU data memory
  - initialize the ICCS interrupt controller
- Enable the IO-Link master channels in PRU

### 7.3.1.2 IOLINK\_close API

The close API performs the following functions when called by the application:

- De-register the hardware and software interrupts
- Disable the IO-Link master channels in PRU

### 7.3.1.3 IOLINK\_control API

The control API performs the following functions when called by the application:

- Control the PRU to send IO-Link master commands to IO-Link devices via PRU firmware (e.g. STARTPULSE, SETCOM, etc.)
- Set the callback functions to the driver which will provide the IO-Link master protocol required timing to the stack when the timer times out and calls back
- Start/stop the software and adjustable timers to provide the required timing
- Control the PRU to set the TX data and send the data to IO-Link devices via PRU firmware

## 7.3.2 IO-Link Master Timers

The IO-Link master stack requires the observation of at least 5 timings per channel. These timings vary in resolution and accuracy and not all of them have to be measured at the same time. Instead of spending 5 hardware timers for every port, this driver implementation uses 2 in total (see below table):

- Cycle timer (100 $\mu$ s timer)
- Adjustable timer (1 $\mu$ s timer resolution)

Timers	Description	Output
100µs Timer	100µs (tick) hardware timer	Provides a channel specific Tcycle (n tick) software timer to the stack. The Timer ISR calls Tcycle software interrupt which in turn calls back to the stack when times out.
		Provides 10ms (100 ticks) software timer to the stack. The Timer ISR calls software timer interrupt which in turn calls back to the stack when times out.
Adjustable Timer	n µsec hardware timer	Provides a high precision (n µsec) timer to the stack. The Timer ISR calls TREN or TDMT software timer interrupt which in turn calls back to the stack when times out.

Table 1: Hardware Timers provided by the driver

The following table shows the timers required by the stack:

Stack Timers	Source	Description
Tcycle	100µs (tick) timer	Channel specific Tcycle timer (n ticks) required by the stack.
TREN	Adjustable timer	Receive enable delay timer
TDMT	Adjustable timer	Master message delay timer
TDWU	10ms software timer derived from 100µs timer	Wake-up retry delay timer
TSD	10ms software timer derived from 100µs timer	Device detection timer

ISDU	10ms software timer derived from 100µs timer	Indexed Service Data Unit timers
------	----------------------------------------------	----------------------------------

Table 2: Stack Timers

### 7.3.3 IO-Link master interrupts

IO-Link master driver serves a lot of time sensitive functionality, which requires deterministic control over the physical layer. It achieves this low latency by using short hardware interrupts and additional software interrupts.

The following figure shows the interrupt driven IO-Link master communication between the stack application and the LLD:



Figure 3 : IO-Link Master Interrupts and Controls

#### 7.3.3.1 Hardware Interrupts

IO-Link master LLD uses three hardware interrupts:

- 100µsec timer interrupt
- Adjustable timer interrupt (1 µsec resolution)
- PRU event interrupt

100µsec Timer interrupt is triggered every 100 µsec (tick), which provides the clock for Tcycle software timer as well as the 10 ms software timer.

Adjustable Timer Interrupt is triggered in n µsec, which provides the high precision time for TREN and TDMT.

PRU event interrupt is triggered by the PRU firmware to report the error or status of a transmission cycle.

### 7.3.3.2 Software Interrupts and callbacks

IO-Link master LLD uses three software interrupts:

- Channel specific Tcycle software interrupt
- 10msec software interrupt
- TREN software timer interrupt
- TDMT software timer interrupt
- Channel specific PRU complete software interrupt

Tcycle software interrupt is triggered by the 100 $\mu$ sec timer hardware ISR when ( $n \times 100\mu\text{s}$ ) time times out, it calls back to the stack to provide the Tcycle timeout event.

10msec software interrupt is triggered by the 100 $\mu$ sec timer hardware ISR when ( $100 \times 100\mu\text{s} = 10\text{msec}$ ) time times out, it calls back to the stack to provide the 10msec tick.

TREN software interrupt is triggered by the adjustable timer hardware ISR when ( $n \times 1\mu\text{s}$ ) time times out, it calls back to the stack to provide the TREN timeout event.

TDMT software interrupt is triggered by the adjustable timer hardware ISR when ( $n \times 1\mu\text{s}$ ) time times out, it calls back to the stack to provide the TDMT timeout event.

PRU complete software interrupt is triggered by the PRU event hardware ISR, it calls back to the stack to provide error or status of a transmission cycle.

### 7.3.4 IO-Link Master Call Flows

#### **IO-LINK Master Open**

Following figure illustrates the sequence for IO-LINK MASTER LLD open operation.

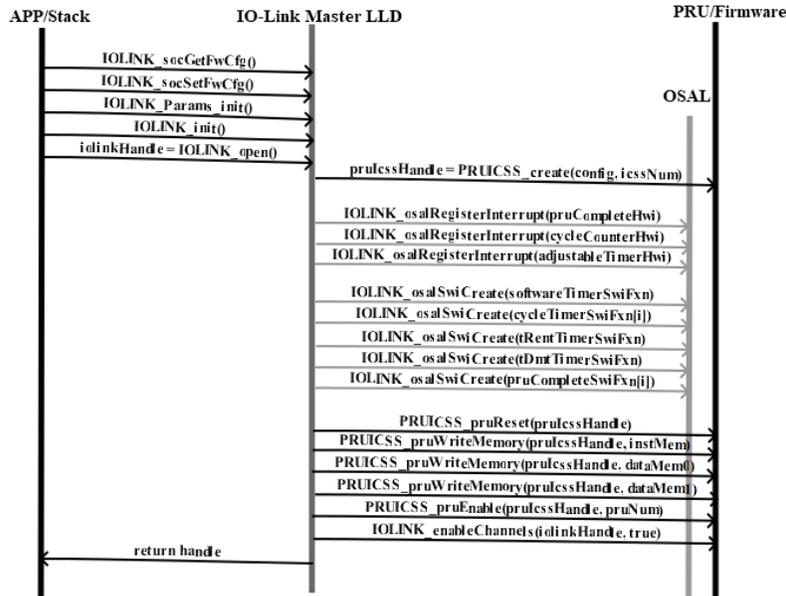


Figure 4 : IO-Link Master Open

**IO-LINK Master Close**

Following figure illustrates the sequence for IO-LINK MASTER LLD close operation.

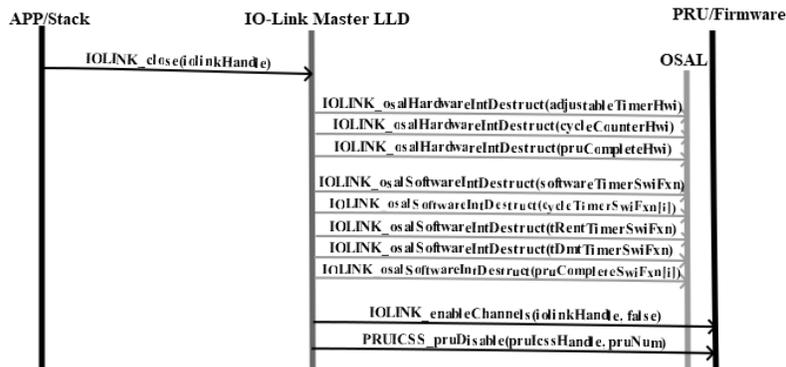


Figure 5 : IO-Link Master Close

**IO-LINK Master Cycle Timer**

Following figure illustrates the sequence for Cycle Timer operation.



**IO-LINK Master 10ms Software Timer**

Following figure illustrates the sequence for 10ms Software Timer operation.

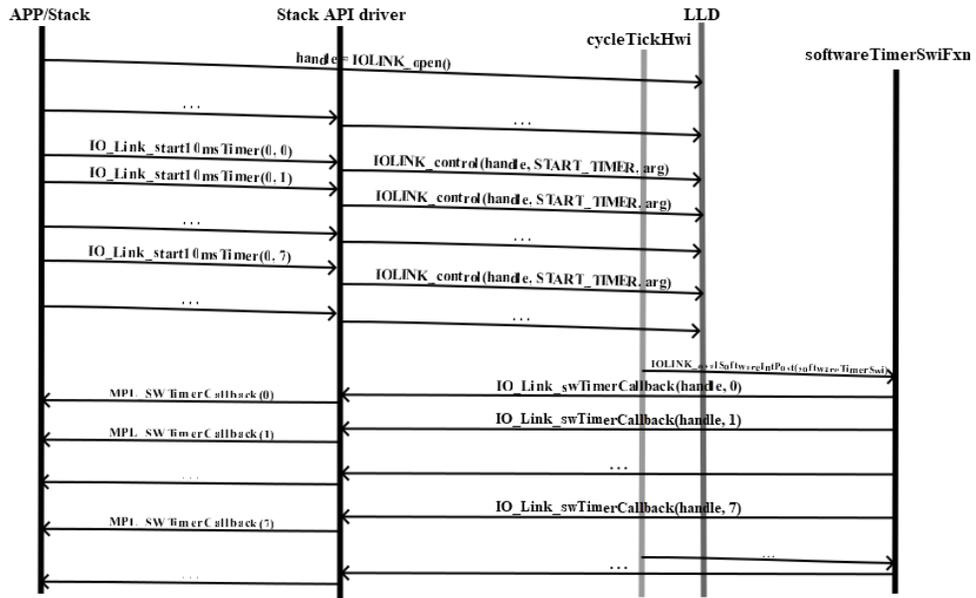


Figure 8 : Software Timer Operation

**IO-LINK Master Wakeup Example**

Following figure illustrates the sequence for IO-Link Master wakeup example.

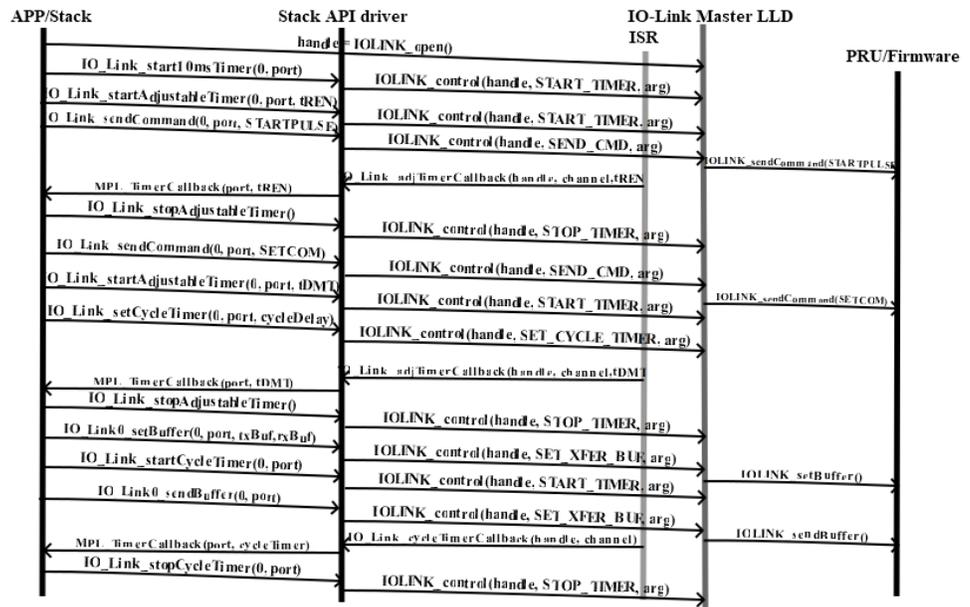
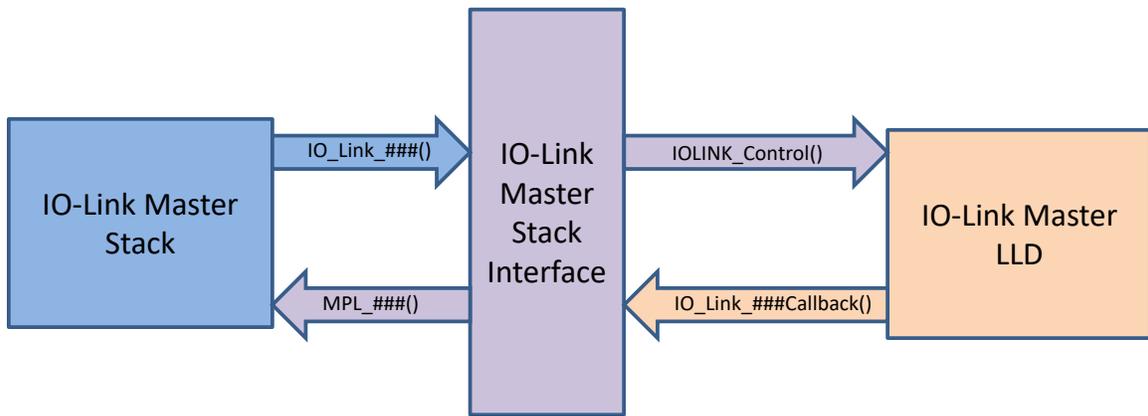


Figure 9 : IO-Link Master Wakeup

**7.4 Integration with IO-Link master stack**

A sample IO-Link master stack interface layer is provided in the test example application, see Figure 2 : IO-Link Master Application Block Diagram. This layer provides an interface between the IO-Link master stack and the LLD.

The stack function calls an interface layer API, in which it calls an LLD API. The LLD callback function calls an interface layer API, in which it calls back to a stack API. The following figure shows the calls between the master stack and LLD via the stack interface:



Color code: TI drivers TI Example 3<sup>rd</sup> party

Figure 10 : IO-Link Master Wakeup

To integrate with a new stack, the below stack interface APIs need to be modified based on the stack APIs:

API/Callback	Description
int8_t IO_Link_sendCommand(uint8_t instance, uint8_t port, uint8_t command, uint8_t arg);	Send user specified command to the LLD for a specified port (channel)
int8_t IO_Link_setBuffer(uint8_t instance, uint8_t port, uint8_t txBufLen, uint8_t rxBufLen, uint8_t *txBuf, uint8_t *rxBuf);	Set the tx and rx buffer for a specified port (channel)

void IO_Link_sendBuffer(uint8_t instance, uint8_t port);	Send the data to LLD for a specified port (channel)
void IO_Link_start10msTimer(uint8_t instance, uint8_t port);	Start the 10 ms timer for a specified port (channel)
void IO_Link_stop10msTimer(uint8_t instance, uint8_t port);	Stop the 10 ms timer for a specified port (channel)
void IO_Link_setCycleTimer(uint8_t instance, uint8_t port, uint32_t delay);	Set the delay time to the cycle timer for a specified port (channel)
void IO_Link_startCycleTimer(uint8_t instance, uint8_t port);	Start the cycle timer for a specified port (channel)
void IO_Link_stopCycleTimer(uint8_t instance, uint8_t port);	Stop the cycle timer for a specified port (channel)
void IO_Link_startAdjustableTimer(uint8_t instance, uint8_t port, uint8_t type, double t);	Start the adjustable timer with a specified delay for a specified timer type and port (channel)
void IO_Link_stopAdjustableTimer(void);	Stop the adjustable timer
void IO_Link_cycleTimerCallback(IOLINK_Handle handle, uint32_t channel);	Cycle timer callback function to the stack for a specified port (channel)
void IO_Link_swTimerCallback(IOLINK_Handle handle, uint32_t channel);	10ms software timer callback function to the stack for a specified port (channel)
void IO_Link_adjTimerCallback(IOLINK_Handle handle, uint32_t channel, uint32_t delayType);	Adjustable timer callback function to the stack for a specified timer type and port (channel)
void IO_Link_xferRspCallback(IOLINK_Handle handle, uint32_t channel);	Data transfer response callback function to the stack for a specified port (channel)
void IO_Link_xferErrRspCallback(IOLINK_Handle	Data transfer error response callback function to the stack for a specified

handle, uint32_t channel);	port (channel)
----------------------------	----------------

Table 3: IO-Link Master Stack Interface APIs

## 7.5 System Resources

Resources like Interrupt, Timers, Memory and internal memory will vary for SoC.

SoC	Interrupt	Timers	Memory	Internal memory
AM437x	100µs timer int: 191 Adj timer int: 127 Pru event int: 192	100µs timer: ICSS0 IEP timer Adj timer: DM Timer 7		

Table 4: Stack Timers

## 8. Test Plan

The test plan should cover the below basic functional tests:

- Example application with stack integrated
  - All the 8 channels (4 channels connected with 4 sensor devices and 4 channels connected with 4 actuator devices)
  - Combination of all the baud rate (com1/com2/com3) depending on the baud rate supported by the device
  - Wakeup sequence
  - Sensor detection (e.g. object presence, input analog signal detection etc.)
  - Actuator controlling (e.g. parameter setting, display/led control, etc.)

The basic functional tests do not cover all the physical layer timing tests.

### 8.1 Hardware

TI provides an IO-Link master reference design board which uses the AM437x IDK and two IO-Link master boards. The two IO-Link master boards have to be stacked together and have to be stacked to the AM437x IDK, refer to 5.

## 8.2 Test Setup

- Connect 8 IO-Link devices (4 Autosens AO001 devices and 4 Autosens AD003 devices) to the IO-Link board via the M8 connectors
- Power on the board
- Load and run the test application on CCS
- The test application should blink the LEDs once the devices are connected to the master successfully and print the following on the UART or CCS console

```
autosens gmbh device AO001 connected on port <port #>
```

```
autosens gmbh device AD003 connected on port <port #>
```

- Put an object close to the AO001 device, the test application should print the following:

```
autosens gmbh device AO001 on port <port #1> detected an  
object, paired device AD003 on <port #2>
```

```
autosens gmbh device AD003 on <port #2> display turned on
```

- Take the object away from the AO001 device, the test application should print the following:

```
autosens gmbh device AO001 on port <port #1> did not detect  
an object, paired device AD003 on port <port #2>
```

```
autosens gmbh device AD003 on <port #2> display turned off
```

- Observe the actuator device which should change the display

## 8.3 Test Automation

System Test should provide a way to automate moving an object close to a sensor device, and monitoring the Display changes on the actuator device.

# 9. Standards, Conventions and Procedures

## 9.1 Documentation Standards

Doxygen format is used for documentation in source code.

## 9.2 Naming conventions

Processor SDK standard naming conventions are used for file and module naming.

## 9.3 Programming Standards

- C99 standard data types are used in driver implementation.

- MISRA-C coding standards are followed wherever applicable

#### 9.4 Software development tools

- TI's Code Composer Studio for project build setup.
- Make files for source code compilation
- Doxygen for extracting documentation from source code
- Klockworks for static code analysis

## 10. System Design

### 10.1 Design Approach

IO-LINK MASTER driver is designed to meet the following requirements:

- Support multiple IO-LINK MASTER instances (if available on the device) per core.
- The driver is OS independent and exposes all the operating system callouts via the OSAL layer.
- Support different versions of IO-LINK MASTER software IP with different versions of IO-Link master stack

Software IP/platform specific functions are mapped to the Software IP/platform independent APIs using function table which is given below

```

/*! Function to open the specified instance */
IOLINK_OpenFxn    openFxn;

/*! Function to close the specified instance */
IOLINK_CloseFxn   closeFxn;

/*! Function to implementation specific control function */
IOLINK_ControlFxn controlFxn;

```

### 10.2 Dependencies

None

## 11. IP Feature List Comparison

This section gives the details of feature comparison of different IO-Link HW IP and software support for those IP features

IO-LINK MASTER IP Features		
	AM437x	
IO-LINK MASTER Mode:	HW	SW
Maximum # of instance supported	NO	1
Maximum # of channels supported per instance	NO	8
COM1/2/3 baud rates	NO	YES
1 start bit generation	NO	YES
1 stop bit generation	NO	YES
8-bit character length	NO	YES
Even parity bit generation and detection	NO	YES

Table 5: IO-Link Master IP Features